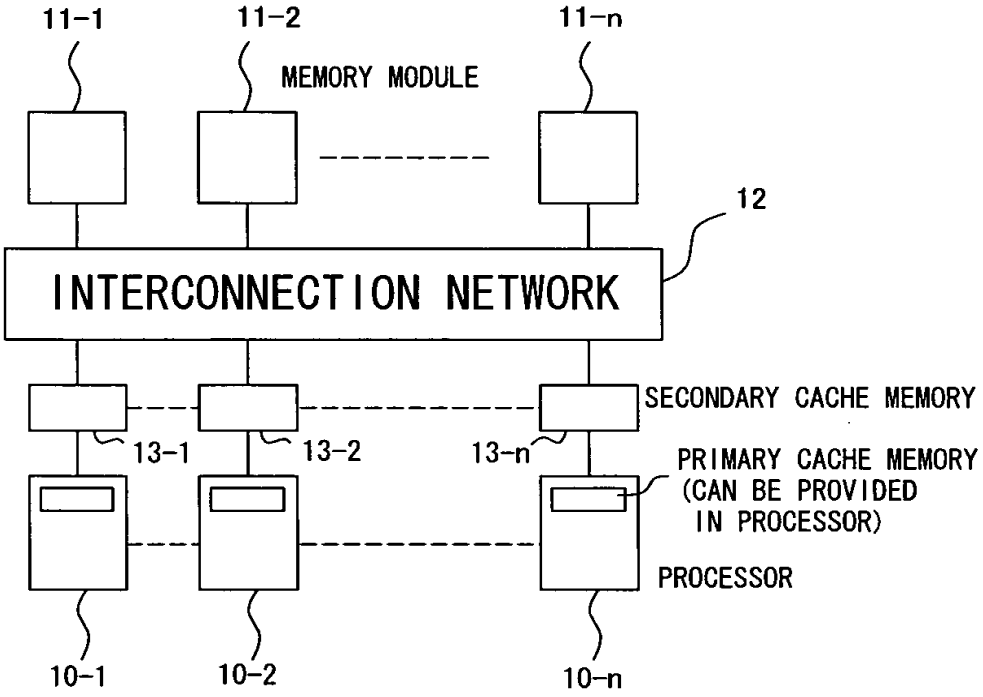

11-1
11-2
11-n
MEMORY MODULE
12
INTERCONNECTION NETWORK
SECONDARY CACHE MEMORY
13-1
13-2
13-n
PRIMARY CACHE MEMORY
(CAN BE PROVIDED
IN PROCESSOR)
PROCESSOR
10-1
10-2
10-n

FIG. 1

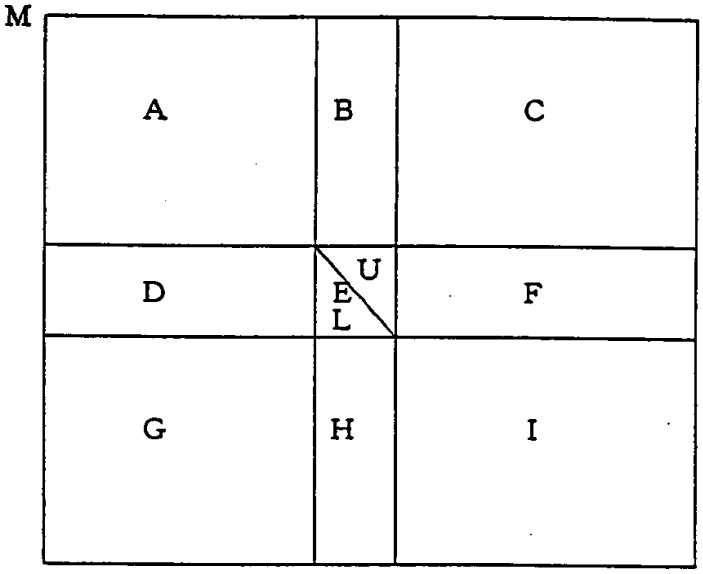


FIG. 2

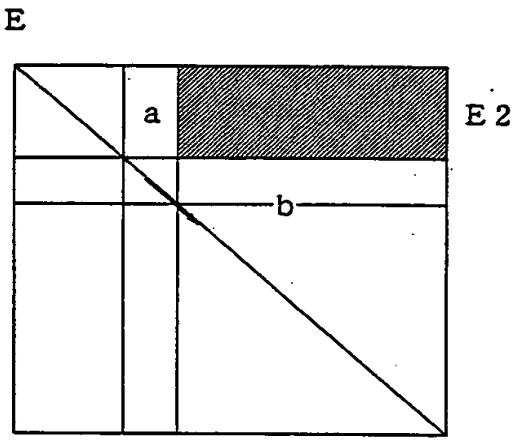


FIG. 3

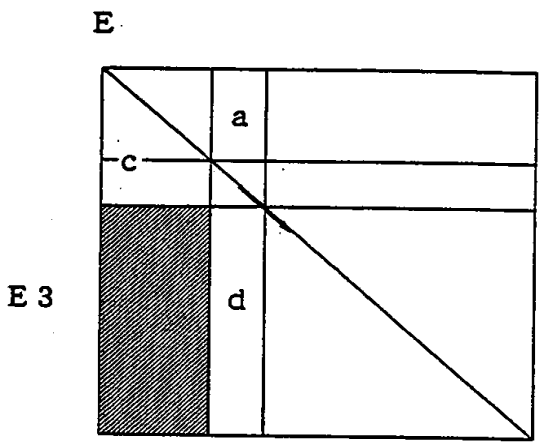


FIG. 4

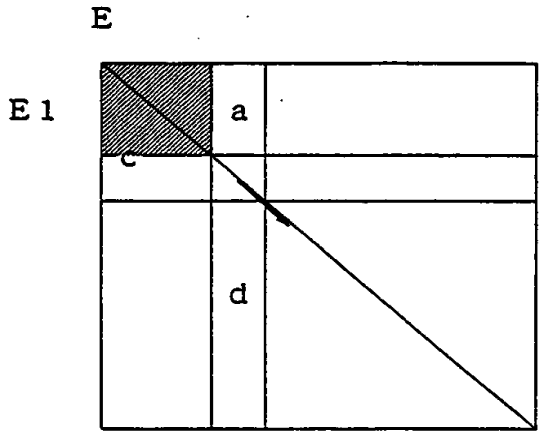


FIG. 5

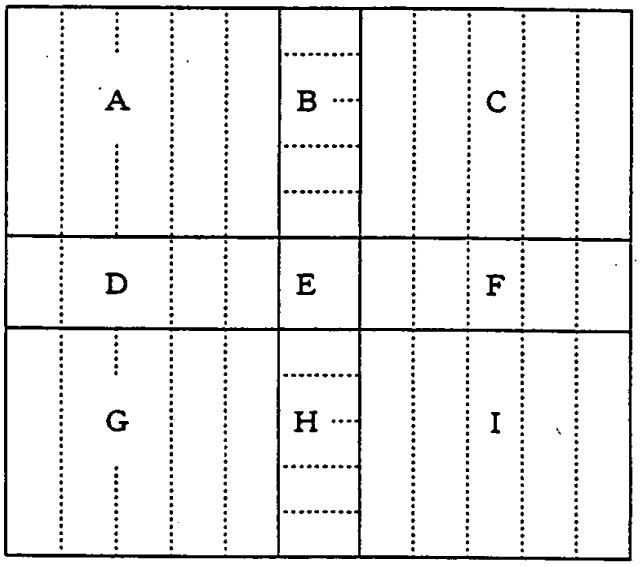


FIG. 6

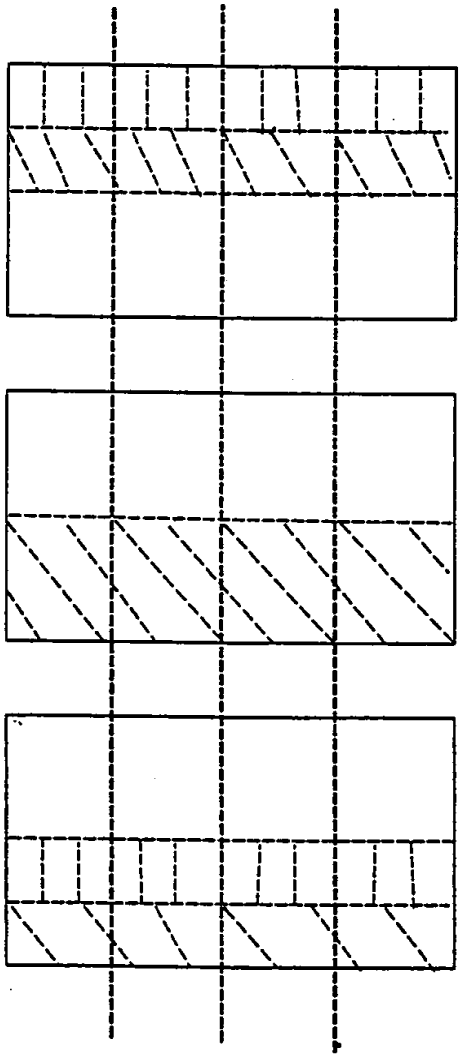


FIG. 7A

FIG. 7C

FIG. 7E

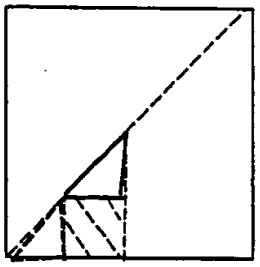


FIG. 7B

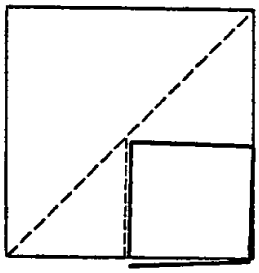


FIG. 7D

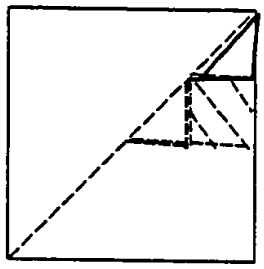


FIG. 7F

E

FIG.8A

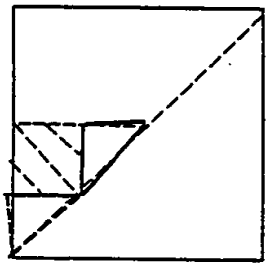


FIG.8B

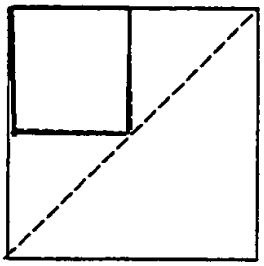
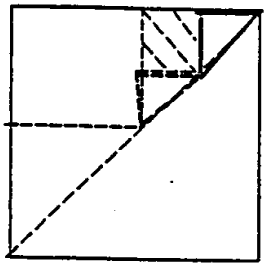
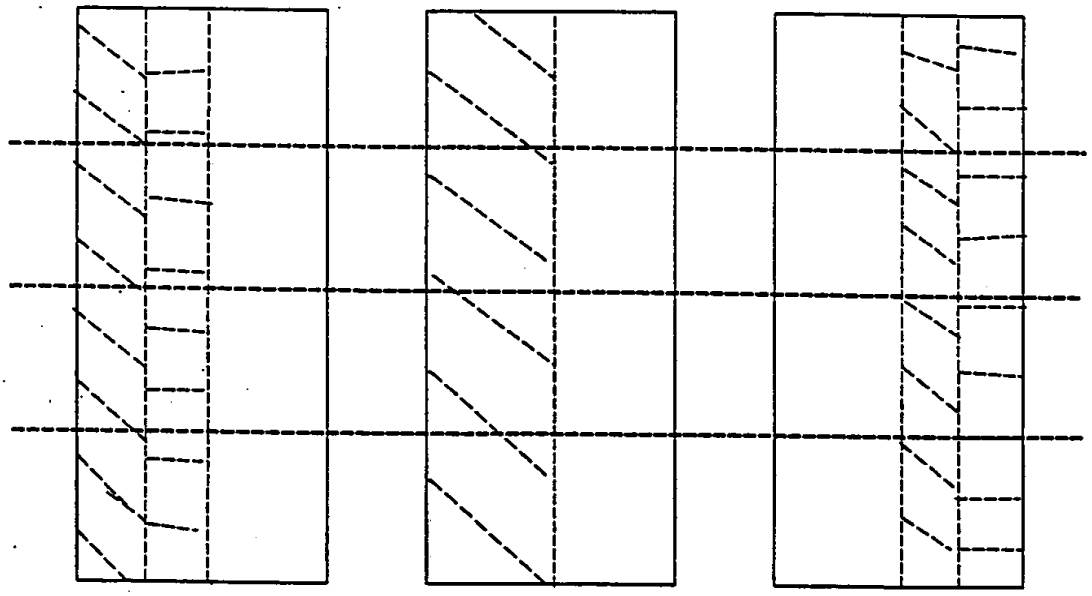


FIG.8C



E



D

```
c     PARALLEL ALGORITHM OF INVERSE MATRIX
      subroutine inversion(a, k, n)
      shared array a(k, n), ip(n)
      create threads
      set nothrd and numthrd
c     nothrd IS THREAD NUMBER 1 ～#TH、numthrd=#TH (TOTAL NUMBER OF THREADS)
      iblk=BLOCK WIDTH
      nb=(n+blk-1)/blk
      do i=1, nb-1
      nbase=(nb-1)*blk
      call LU(a, k, n, nbase, blk, ip, nothrd, numthrd)
                ! LU DECOMPOSITION IS PERFORMED ON SPECIFIED BLOCK, ROW BLOCK IS
UPDATED,
                ! OPERATION FOR UPDATING LOWER RIGHT SQUARE BLOCK MATRIX ARE PERFORMED
                ! IN PARALLEL. ROW INTERCHANGING INFORMATION IS RETURNED TO
                  ip(nbase+1:nbase-blk).
                ! FOR PARALLEL ALGORITHM OF THIS PORTION, REFER TO JAPANESE PATENT
                  APPLICATION NO. HEI-12-358232
      do i=nbase+1, nbase+blk
      if(ip(i)>i)then
      exchange(a(i, 1:nbase), a(ip(i), 1:nbase))
      endif
      enddo
      call update(a, k, n, nbase, blk, nothrd, numthrd)
      enddo
      nbase=(nb-1)*blk
      blklast=n-nbase
      call LU(a, k, n, nbase, blklast, nothrd, numthrd)
      call update(a, k, n, nbase, blklast, nothrd, numthrd)
      return
      end
```

FIG. 9

```
c     ROUTINE OF UPDATING REMAINING PORTION ACCORDING TO INFORMATION
      ABOUT BLOCK LU DECOMPOSITION
      subrourine update(a, k, n, nbase, blk, nothrd, numthrd)
      call b-update(a, k, n, nbase, blk, nothrd, numthrd)
      call d-update(a, k, n, nbase, blk, nothrd, numthrd)
      call c-update(a, k, n, nbase, blk, nothrd, numthrd)
      call a-update(a, k, n, nbase, blk, nothrd, numthrd)
      call g-update(a, k, n, nbase, blk, nothrd, numthrd)
      BARRIER SYNC
      if(nothread=1)then
c     UPDATE 1 OF e
      call e-update1(a(nbase+1, nbase+1), k, n, blk)
      endif
      BARRIER SYNC
      len←(nbase+numthrd-1)/numthrd
      is←(nothrd-1)*len+1
      ie←nothrd*len
      call df-update(a(nbase+1, 1), k, n, a(nbase+1, nbase+1), is, ie, blk)
      nbase2←nbase+blk
      len←(n-nbase2+numthrd-1)/numthrd
      is2←nbase2+(nothrd-1)*len+1
      ie2←nbase2+nothrd*len
      call df-update(a(nbase+1, 1), k, n, a(nbase+1, nbase+1), is, ie, blk)
      BARRIER SYNC
      if(nothread=1)then
c     UPDATE 2 OF e
      call e-update2(a(nbase+1, nbase+1), k, n, blk)
      endif
      BARRIER SYNC
      len←(nbase+numthrd-1)/numthrd
      is←(nothrd-1)*len+1
      ie←nothrd*len
      call bh-update(a(1, nbase+1), k, n, a(nbase+1, nbase+1), is, ie, blk)
      nbase2←nbase+blk
      len←(n-nbase2+numthrd-1)/numthrd
      is2←nbase2+(nothrd-1)*len+1
      ie2←nbase2+nothrd*len
      call bh-update(a(1, nbase+1), k, n, a(nbase+1, nbase+1), is, ie, blk)
      BARRIER SYNC
      if(nothread=1)then
c     UPDATE 3 OF e
      call e-update3(a(nbase+1, nbase+1), k, n, blk)
      endif
      BARRIER SYNC
      len=(n+numthrd-1)/numthrd
      is=(nothrd-1)*len+1
      ie=min(n, nothrd*len)
      if(ip(i)>i)then
      exchange(a(is:ie, i), a(is:ie, ip(i)))
      endif
      enddo
      BARRIER SYNC

      eliminate threads

      return
      end
```

FIG. 10

```
c     UPDATE OF BLOCK B
      subroutine b-update(a, k, n, nbase, blk, nothrd, numthrd)
      shared array a(k, n)
      len←(nbase+numthrd-1)/numthrd
      is1←(nothrd-1)*len
      ie1←nothrd*len
      a(is:ie, iof+1:iof+blk)
      ←a(is:ie, iof+1:iof+blk)*TRU-U(iof+1:iof+blk, iof+1:iof+blk)-1
        ! TRU-U  UPPER TRIANGULAR MATRIX WITH DIAGONAL ELEMENT=1.0
      return
      end

c     UPDATE OF BLOCK D
      subroutine d-update(a, k, n, nbase, blk, nothrd, numthrd)
      shared array a(k, n)
      len←(nbase+numthrd-1)/numthrd
      is1←(nothrd-1)*len
      ie1←nothrd*lenis1
      iof=nbase
      a(is:ie, iof+1:iof+blk)
      ←TRL(iof+1:iof+blk, iof+1:iof+blk)-1*a(is:ie, iof+1:iof+blk)
           ! TRL INDICATES LOWER TRIANGULAR MATRIX
      return
      end
```

FIG. 11

```
c     UPDATE OF BLOCK C
      subroutine c-update(a, k, n, nbase, blk, nothrd, numthrd)
      shared array a(k, n)
      nbase2←nbase+blk
      iof=nbase
      len←(n-nbase2+numthrd-1)/numthrd
      is2←nbase2+(nothrd-1)*len+1
      ie2←nbase2+nothrd*len
      a(1:iof, is2:ie2)
      ←a(1:iof, is2:ie2)
      -a(1:iof, iof+1:iof+blk)*a(iof+1:iof+blk, is2:ie2)
      return
      end

c     UPDATE OF BLOCK A
      subroutine a-update(a, k, n, nbase, blk, nothrd, numthrd)
      shared array a(k, n)
      len←(nbase+numthrd-1)/numthrd
      is2←(nothrd-1)*len
      ie2←nothrd*len
      iof=nbase
      a(1:iof, is2:ie2)
      ←a(1:iof, is2:ie2)
      -a(1:iof, iof+1:iof+blk)*a(iof+1:iof+blk, is2:ie2)
      return
      end
```

FIG. 12

```
c     UPDATE OF BLOCK G
      subroutine g-update(a, k, n, nbase, blk, nothrd, numthrd)
      shared array a(k, n)
      len←(nbase+numthrd-1)/numthrd
      is2←(nothrd-1)*len
      ie2←nothrd*len
      iof←nbase+blk
      a(iof+1:n, is2:ie2)
      ←a(iof+1:n, is2:ie2)
      -a(iof+1:n, iof+1:iof+blk)*a(iof+1:iof+blk, is2:ie2)
      return
      end

c     FIRST UPDATE OF BLOCK E
      subroutine e-update1(s, k, n, blk)
      shared array s(k, n)

      do i=1, blk
      s(1:i-1, i+1:blk)←s(1:i-1, i+1:blk)-s(1:i-1, i)*s(I, i+1:blk)
      enddo
      return
      end

c     SECOND UPDATE OF BLOCK E
      subroutine e-update2(s, k, n, blk)
      shared array s(k, n)

      do i=1, blk
      tmp←1.0/s(i, i)
      s(i, 1:i-1)←tmp*s(i, 1:i-1)
      s(i+1:blk, 1:i-1)←s(i+1:blk, 1:i-1)-s(i, 1:i-1)*s(i+1:blk, i)
      s(i+1:blk, i)←-s(i+1:blk, i)*tmp
      s(i, i)←tmp
      enddo
      return
      end
```

FIG. 13

```
c     FINAL UPDATE OF BLOCK E
      subriutine e-update3(s, k, n, blk)
      shared array s(k, n)

      do i=1, blk
      s(1:i-1, 1:i-1)←s(1:i-1, 1:i-1)-s(1:i-1, i)*s(i, 1:i-1)
      s(1:i-1, i)←-s(1:i-1, i)*a(i, i)
      enddo
      return
      end

c     UPDATE OF BLOCKS D AND F
      subroutine df-update(a, k, n, s, is, ie, len)
      shared array a(k, *), s(k, *)
      if(len<10)then
      do i=1, len
      a(1:i-1, is:ie)←a(1:i-1, is:ie)-s(1:i-1, i)*a(i, is:ie)
      enddo
      else
      if(len>=32 or len<=20)then
      len1←len/2
      len2←len-len1
      else
      len1←len/3
      len2←len-len1
      endif
      call df-update(a, k, n, s, is, ie, len1)
      a(1:len1, is:ie)
      ←a(1:len1, is:ie)-s(1:len1, len1+1:len)*a(len1+1:len, is:ie)
      call df-update(a(len1+1, 1), k, n, s(len1+1, len1+1), is, ie, len2)
      endif
      return
      end
```

FIG. 14

```
c     UPDATE OF BLOCKS B AND H
      subroutine bh-update(a, k, n, s, is, ie, len)
      shared array a(k, *), s(k, *)
      if(len<10)then
      do i=1, len
      a(is:ie, i)←-a(is:ie, i)*s(i, i)
      a(is:ie, i)←a(is:ie, i)-a(is:ie, i+1:len)*s(i+1:len, i)
      enddo
      else
      if(len>=32 or len<=20)then
      len1←len/2
      len2←len-len1
      else
      len1←len/3
      len2←len-len1
      endif
      call bh-update(a, k, n, s, is, ie, len1)
      a(is:ie, 1:len1)←a(is:ie, 1:len1)
                      -a(is:ie, len1+1:len)*s(len1+1:len, 1:len1)
      call bh-update(a(1, len1+1), k, n, s(len1+1, len1+1), is, ie, len2)
      endif
      return
      end
```

FIG. 15

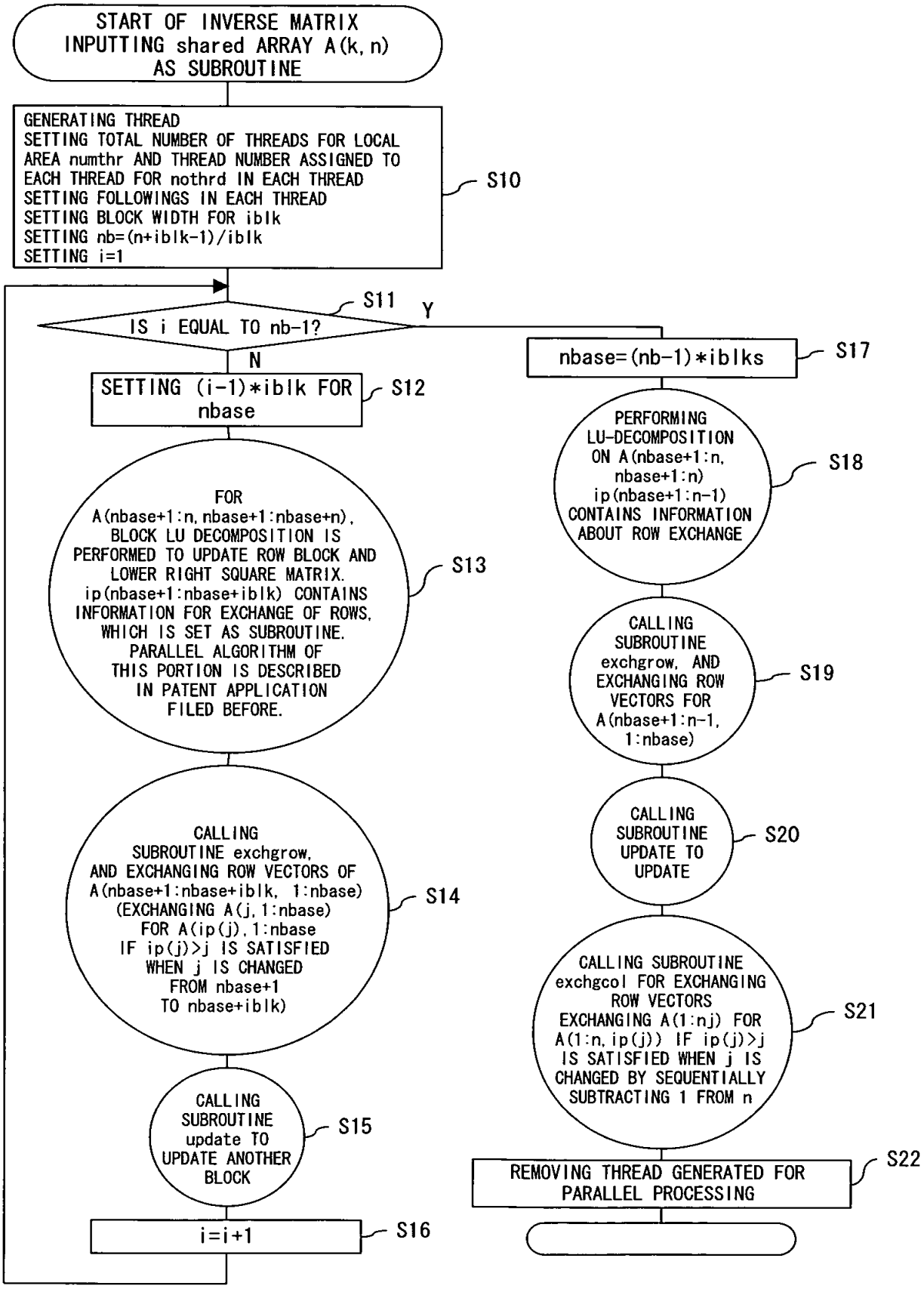

START OF INVERSE MATRIX
INPUTTING shared ARRAY A(k, n)
AS SUBROUTINE
GENERATING THREAD
SETTING TOTAL NUMBER OF THREADS FOR LOCAL AREA numthr AND THREAD NUMBER ASSIGNED TO EACH THREAD FOR nothrd IN EACH THREAD
SETTING FOLLOWINGS IN EACH THREAD
SETTING BLOCK WIDTH FOR iblk
SETTING nb=(n+iblk-1)/iblk
SETTING i=1
S10
IS i EQUAL TO nb-1?
S11
Y
N
SETTING (i-1)*iblk FOR nbase
S12
FOR A(nbase+1:n, nbase+1:nbase+n), BLOCK LU DECOMPOSITION IS PERFORMED TO UPDATE ROW BLOCK AND LOWER RIGHT SQUARE MATRIX. ip(nbase+1:nbase+iblk) CONTAINS INFORMATION FOR EXCHANGE OF ROWS, WHICH IS SET AS SUBROUTINE. PARALLEL ALGORITHM OF THIS PORTION IS DESCRIBED IN PATENT APPLICATION FILED BEFORE.
S13
CALLING SUBROUTINE exchgrow, AND EXCHANGING ROW VECTORS OF A(nbase+1:nbase+iblk, 1:nbase) (EXCHANGING A(j, 1:nbase) FOR A(ip(j), 1:nbase IF ip(j)>j IS SATISFIED WHEN j IS CHANGED FROM nbase+1 TO nbase+iblk)
S14
CALLING SUBROUTINE update TO UPDATE ANOTHER BLOCK
S15
i=i+1
S16
nbase=(nb-1)*iblks
S17
PERFORMING LU-DECOMPOSITION ON A(nbase+1:n, nbase+1:n) ip(nbase+1:n-1) CONTAINS INFORMATION ABOUT ROW EXCHANGE
S18
CALLING SUBROUTINE exchgrow, AND EXCHANGING ROW VECTORS FOR A(nbase+1:n-1, 1:nbase)
S19
CALLING SUBROUTINE UPDATE TO UPDATE
S20
CALLING SUBROUTINE exchgcol FOR EXCHANGING ROW VECTORS EXCHANGING A(1:nj) FOR A(1:n, ip(j)) IF ip(j)>j IS SATISFIED WHEN j IS CHANGED BY SEQUENTIALLY SUBTRACTING 1 FROM n
S21
REMOVING THREAD GENERATED FOR PARALLEL PROCESSING
S22

FIG. 16

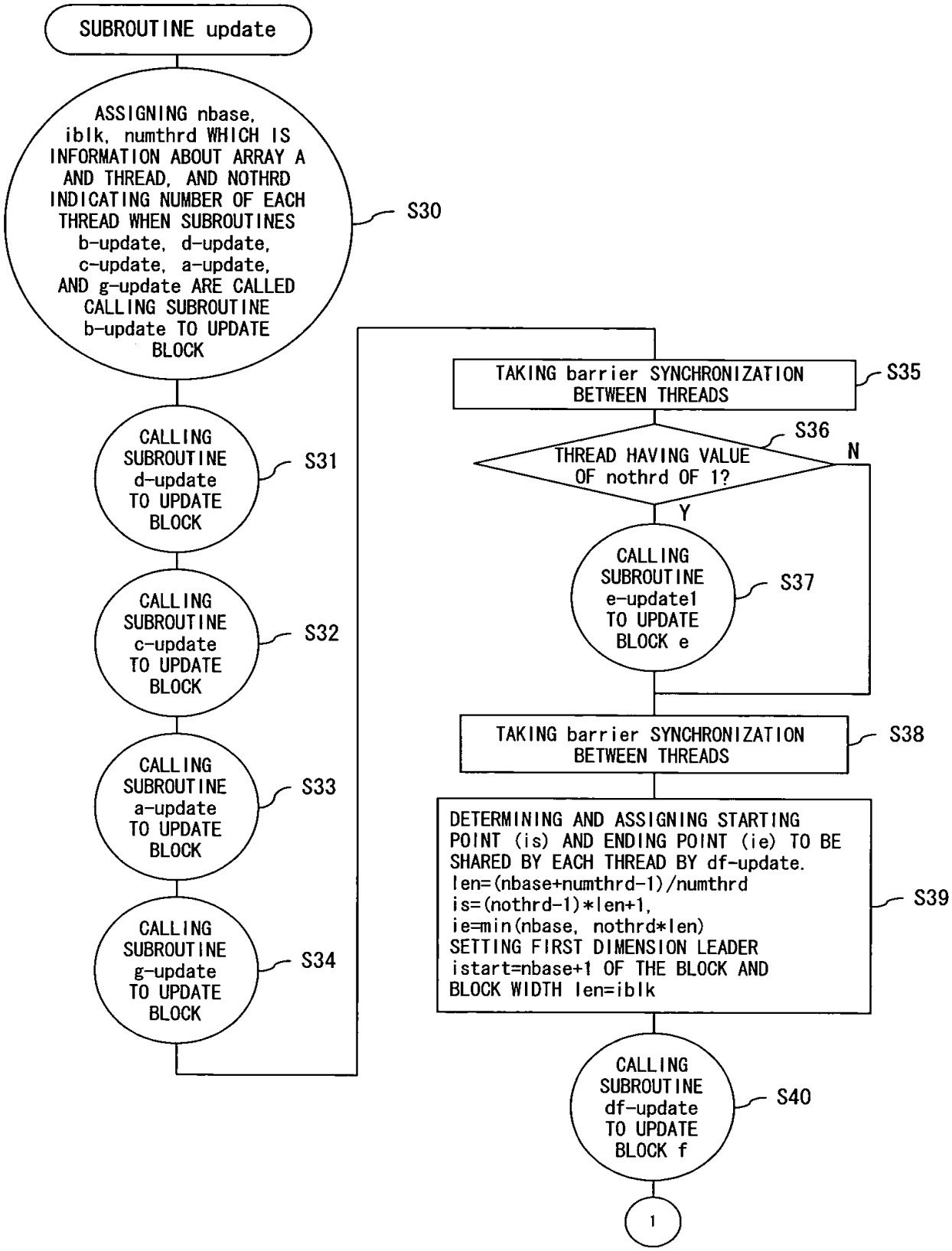

SUBROUTINE update
ASSIGNING nbase, iblk, numthrd WHICH IS INFORMATION ABOUT ARRAY A AND THREAD, AND NOTHRD INDICATING NUMBER OF EACH THREAD WHEN SUBROUTINES b-update, d-update, c-update, a-update, AND g-update ARE CALLED CALLING SUBROUTINE b-update TO UPDATE BLOCK
S30
CALLING SUBROUTINE d-update TO UPDATE BLOCK
S31
CALLING SUBROUTINE c-update TO UPDATE BLOCK
S32
CALLING SUBROUTINE a-update TO UPDATE BLOCK
S33
CALLING SUBROUTINE g-update TO UPDATE BLOCK
S34
TAKING barrier SYNCHRONIZATION BETWEEN THREADS
S35
THREAD HAVING VALUE OF nothrd OF 1?
S36
N
Y
CALLING SUBROUTINE e-update1 TO UPDATE BLOCK e
S37
TAKING barrier SYNCHRONIZATION BETWEEN THREADS
S38
DETERMINING AND ASSIGNING STARTING POINT (is) AND ENDING POINT (ie) TO BE SHARED BY EACH THREAD BY df-update.
len=(nbase+numthrd-1)/numthrd
is=(nothrd-1)*len+1,
ie=min(nbase, nothrd*len)
SETTING FIRST DIMENSION LEADER istart=nbase+1 OF THE BLOCK AND BLOCK WIDTH len=iblk
S39
CALLING SUBROUTINE df-update TO UPDATE BLOCK f
S40
1

FIG. 17

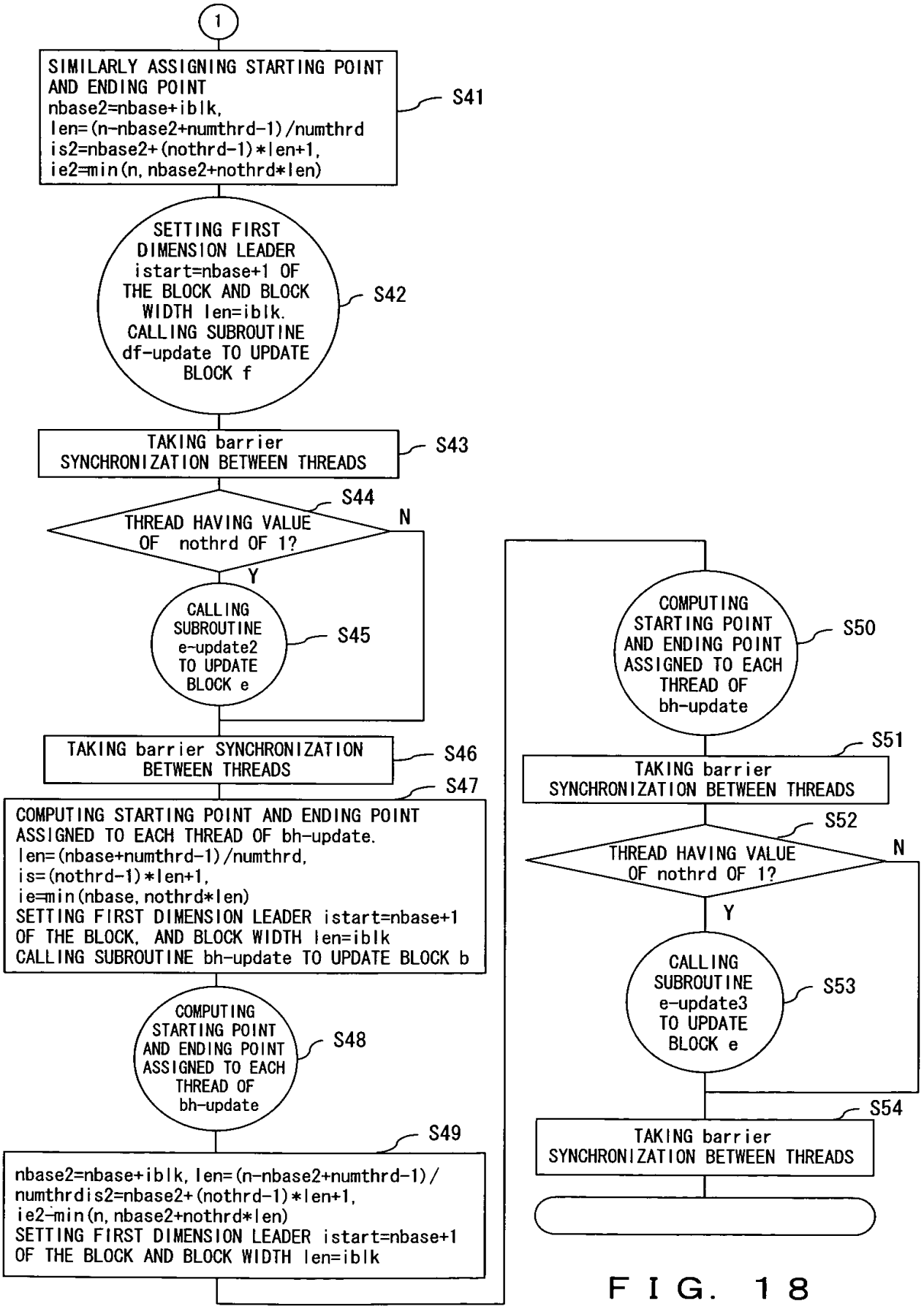
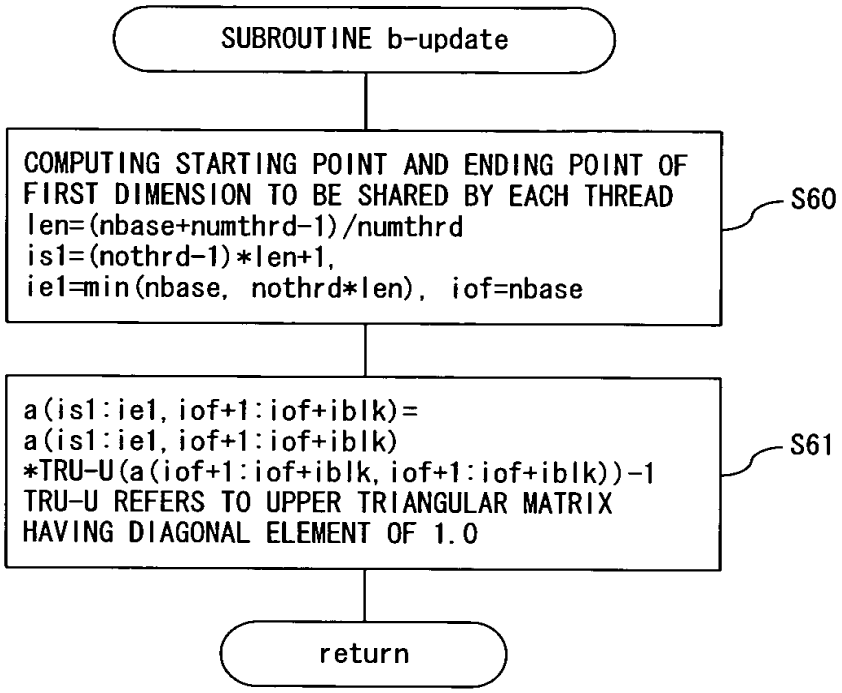

1
SIMILARLY ASSIGNING STARTING POINT AND ENDING POINT
nbase2=nbase+iblk,
len=(n-nbase2+numthrd-1)/numthrd
is2=nbase2+(nothrd-1)*len+1,
ie2=min(n, nbase2+nothrd*len)
S41
SETTING FIRST DIMENSION LEADER istart=nbase+1 OF THE BLOCK AND BLOCK WIDTH len=iblk. CALLING SUBROUTINE df-update TO UPDATE BLOCK f
S42
TAKING barrier SYNCHRONIZATION BETWEEN THREADS
S43
THREAD HAVING VALUE OF nothrd OF 1?
S44
N
Y
CALLING SUBROUTINE e-update2 TO UPDATE BLOCK e
S45
TAKING barrier SYNCHRONIZATION BETWEEN THREADS
S46
S47
COMPUTING STARTING POINT AND ENDING POINT ASSIGNED TO EACH THREAD OF bh-update.
len=(nbase+numthrd-1)/numthrd,
is=(nothrd-1)*len+1,
ie=min(nbase, nothrd*len)
SETTING FIRST DIMENSION LEADER istart=nbase+1 OF THE BLOCK, AND BLOCK WIDTH len=iblk
CALLING SUBROUTINE bh-update TO UPDATE BLOCK b
COMPUTING STARTING POINT AND ENDING POINT ASSIGNED TO EACH THREAD OF bh-update
S48
S49
nbase2=nbase+iblk, len=(n-nbase2+numthrd-1)/numthrdis2=nbase2+(nothrd-1)*len+1,
ie2-min(n, nbase2+nothrd*len)
SETTING FIRST DIMENSION LEADER istart=nbase+1 OF THE BLOCK AND BLOCK WIDTH len=iblk
COMPUTING STARTING POINT AND ENDING POINT ASSIGNED TO EACH THREAD OF bh-update
S50
S51
TAKING barrier SYNCHRONIZATION BETWEEN THREADS
S52
THREAD HAVING VALUE OF nothrd OF 1?
N
Y
CALLING SUBROUTINE e-update3 TO UPDATE BLOCK e
S53
S54
TAKING barrier SYNCHRONIZATION BETWEEN THREADS

FIG. 18


SUBROUTINE b-update
COMPUTING STARTING POINT AND ENDING POINT OF FIRST DIMENSION TO BE SHARED BY EACH THREAD
len=(nbase+numthrd-1)/numthrd
is1=(nothrd-1)*len+1,
ie1=min(nbase, nothrd*len), iof=nbase
S60
a(is1:ie1, iof+1:iof+iblk)=
a(is1:ie1, iof+1:iof+iblk)
*TRU-U(a(iof+1:iof+iblk, iof+1:iof+iblk))-1
TRU-U REFERS TO UPPER TRIANGULAR MATRIX HAVING DIAGONAL ELEMENT OF 1.0
S61
return

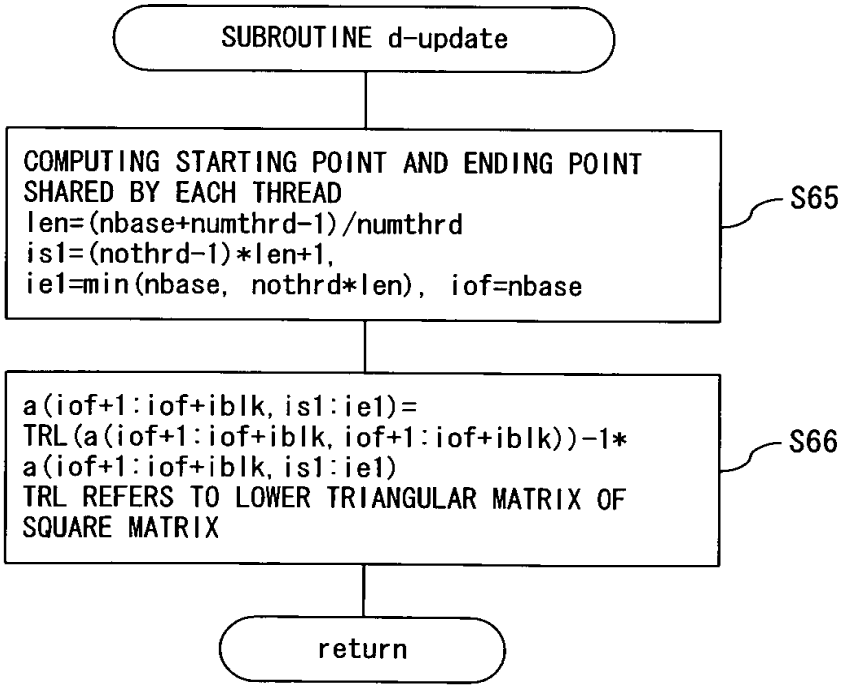

SUBROUTINE d-update
COMPUTING STARTING POINT AND ENDING POINT SHARED BY EACH THREAD
len=(nbase+numthrd-1)/numthrd
is1=(nothrd-1)*len+1,
ie1=min(nbase, nothrd*len), iof=nbase
S65
a(iof+1:iof+iblk, is1:ie1)=
TRL(a(iof+1:iof+iblk, iof+1:iof+iblk))-1*
a(iof+1:iof+iblk, is1:ie1)
TRL REFERS TO LOWER TRIANGULAR MATRIX OF SQUARE MATRIX
S66
return

FIG. 19

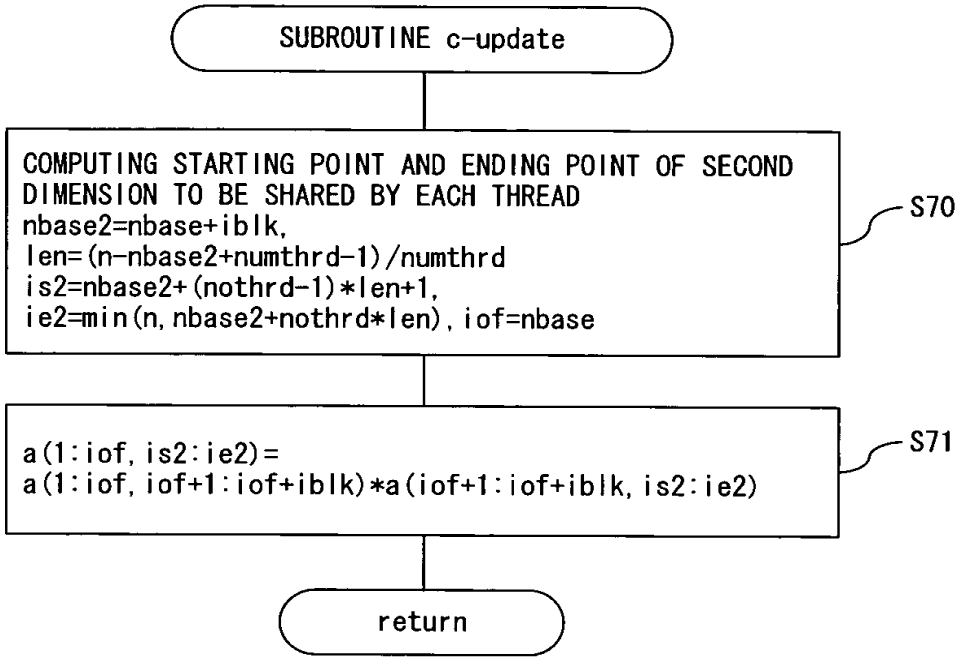

SUBROUTINE c-update
COMPUTING STARTING POINT AND ENDING POINT OF SECOND DIMENSION TO BE SHARED BY EACH THREAD
nbase2=nbase+iblk,
len=(n-nbase2+numthrd-1)/numthrd
is2=nbase2+(nothrd-1)*len+1,
ie2=min(n, nbase2+nothrd*len), iof=nbase
S70
a(1:iof, is2:ie2)=
a(1:iof, iof+1:iof+iblk)*a(iof+1:iof+iblk, is2:ie2)
S71
return

FIG. 20

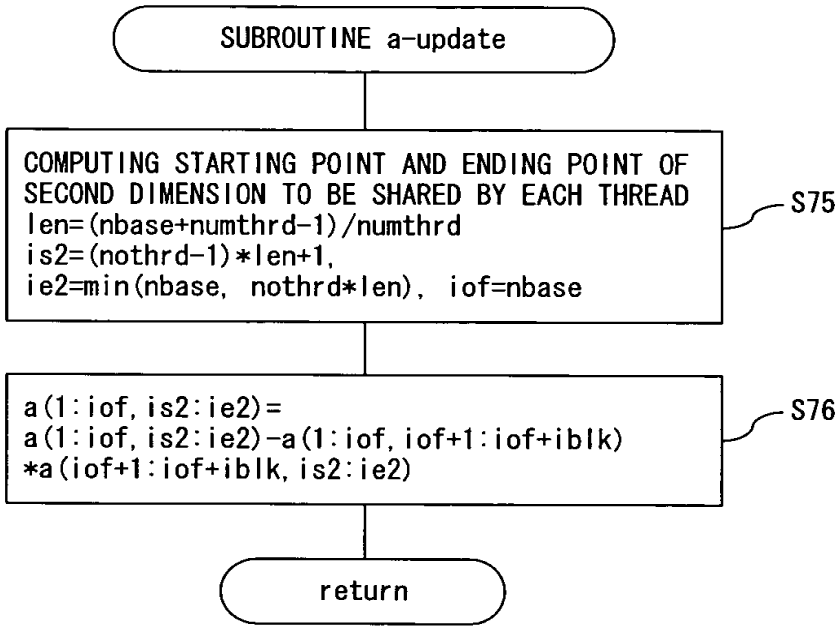

SUBROUTINE a-update
COMPUTING STARTING POINT AND ENDING POINT OF SECOND DIMENSION TO BE SHARED BY EACH THREAD
len=(nbase+numthrd-1)/numthrd
is2=(nothrd-1)*len+1,
ie2=min(nbase, nothrd*len), iof=nbase
S75
a(1:iof, is2:ie2)=
a(1:iof, is2:ie2)-a(1:iof, iof+1:iof+iblk)
*a(iof+1:iof+iblk, is2:ie2)
S76
return

FIG. 21

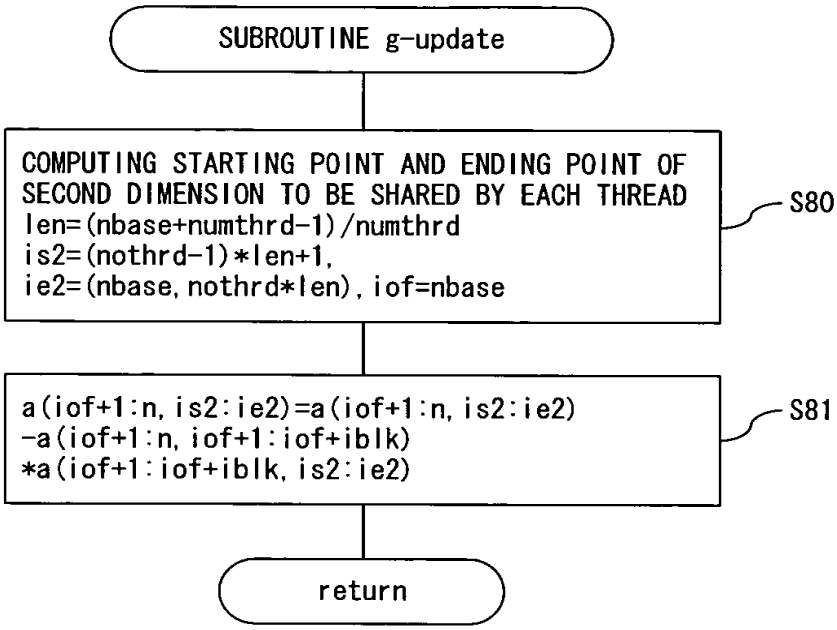

SUBROUTINE g-update
COMPUTING STARTING POINT AND ENDING POINT OF SECOND DIMENSION TO BE SHARED BY EACH THREAD
len=(nbase+numthrd-1)/numthrd
is2=(nothrd-1)*len+1,
ie2=(nbase, nothrd*len), iof=nbase
S80
a(iof+1:n, is2:ie2)=a(iof+1:n, is2:ie2)
-a(iof+1:n, iof+1:iof+iblk)
*a(iof+1:iof+iblk, is2:ie2)
S81
return

FIG. 22

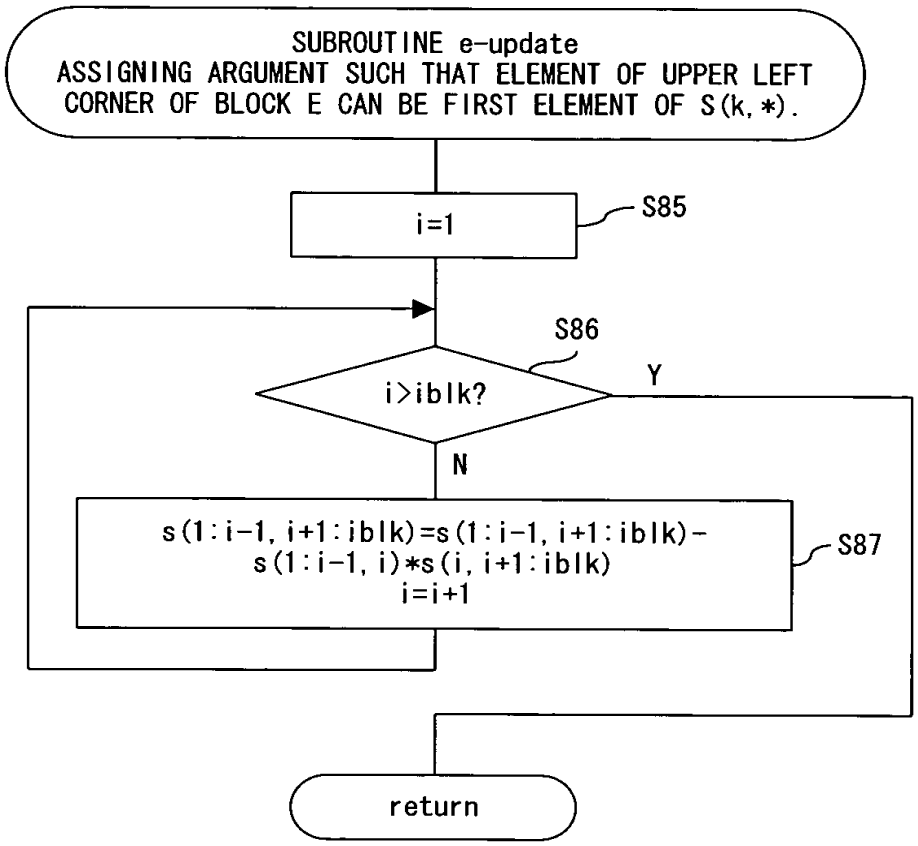

SUBROUTINE e-update
ASSIGNING ARGUMENT SUCH THAT ELEMENT OF UPPER LEFT CORNER OF BLOCK E CAN BE FIRST ELEMENT OF S(k,*).
i=1
S85
S86
i>iblk?
Y
N
s(1:i-1, i+1:iblk)=s(1:i-1, i+1:iblk)-
s(1:i-1, i)*s(i, i+1:iblk)
i=i+1
S87
return

FIG. 23

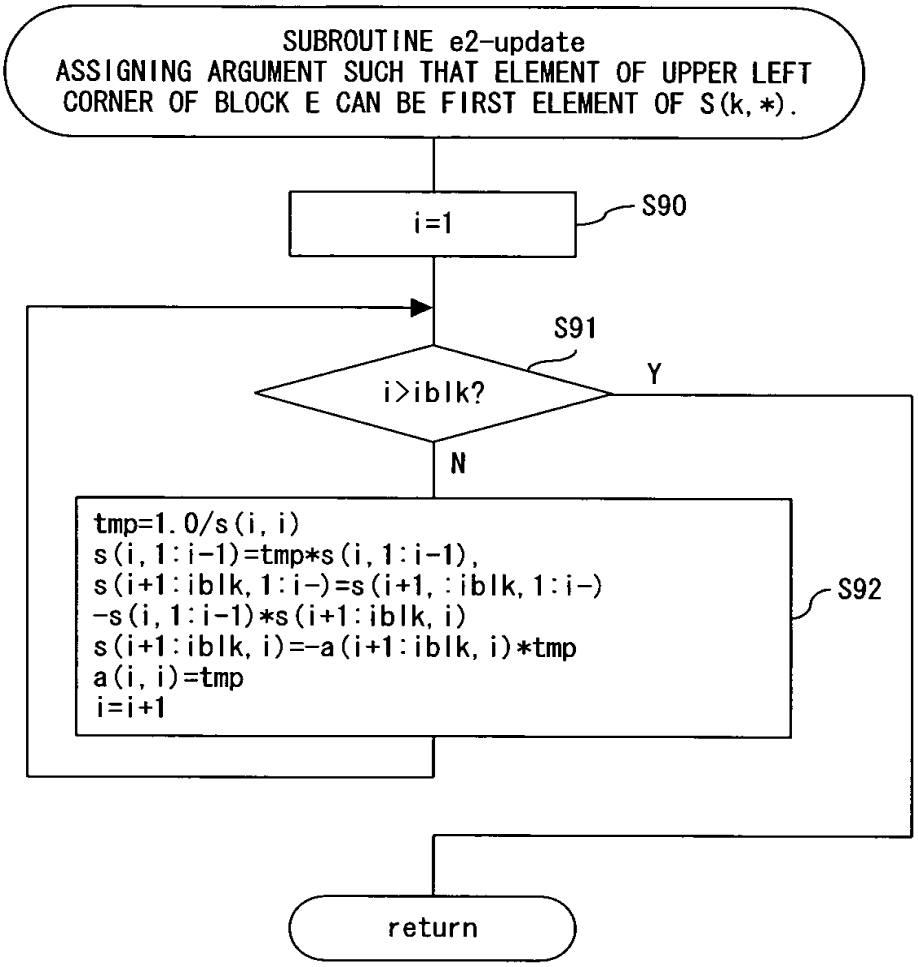

SUBROUTINE e2-update
ASSIGNING ARGUMENT SUCH THAT ELEMENT OF UPPER LEFT CORNER OF BLOCK E CAN BE FIRST ELEMENT OF S(k,*).
i=1
S90
S91
i>iblk?
Y
N
tmp=1.0/s(i,i)
s(i,1:i-1)=tmp*s(i,1:i-1),
s(i+1:iblk,1:i-)=s(i+1,:iblk,1:i-)
-s(i,1:i-1)*s(i+1:iblk,i)
s(i+1:iblk,i)=-a(i+1:iblk,i)*tmp
a(i,i)=tmp
i=i+1
S92
return

FIG. 24

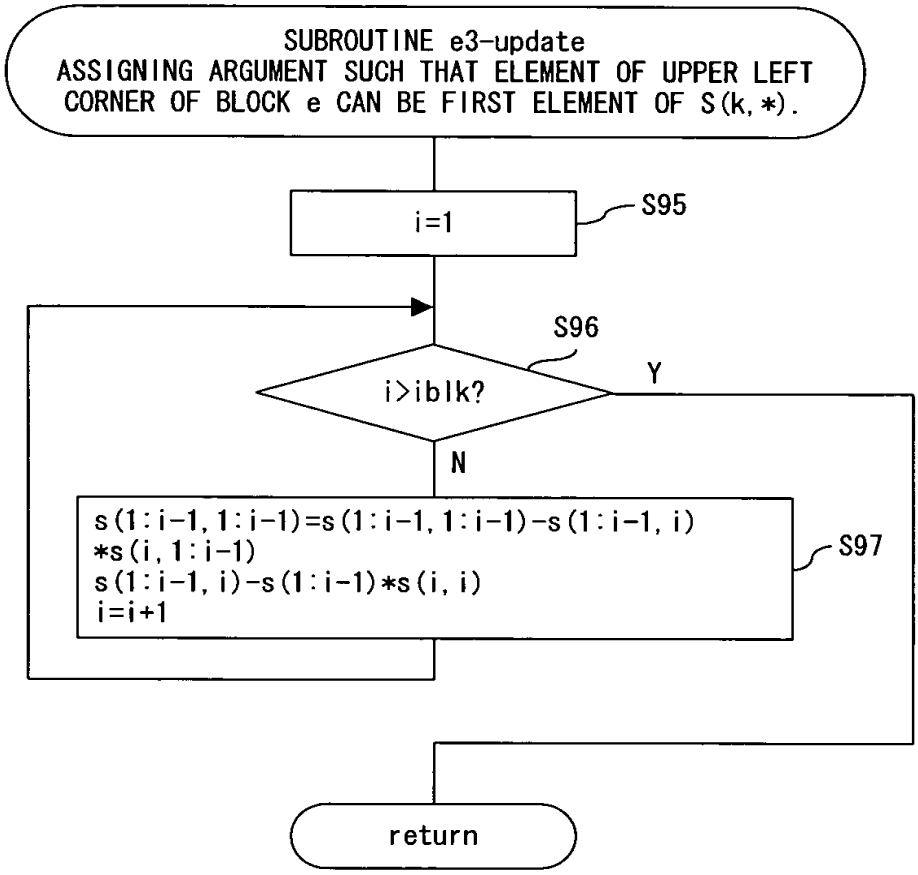

SUBROUTINE e3-update
ASSIGNING ARGUMENT SUCH THAT ELEMENT OF UPPER LEFT CORNER OF BLOCK e CAN BE FIRST ELEMENT OF S(k,*).
i=1
S95
S96
i>iblk?
Y
N
s(1:i-1,1:i-1)=s(1:i-1,1:i-1)-s(1:i-1,i)
*s(i,1:i-1)
s(1:i-1,i)-s(1:i-1)*s(i,i)
i=i+1
S97
return

FIG. 25

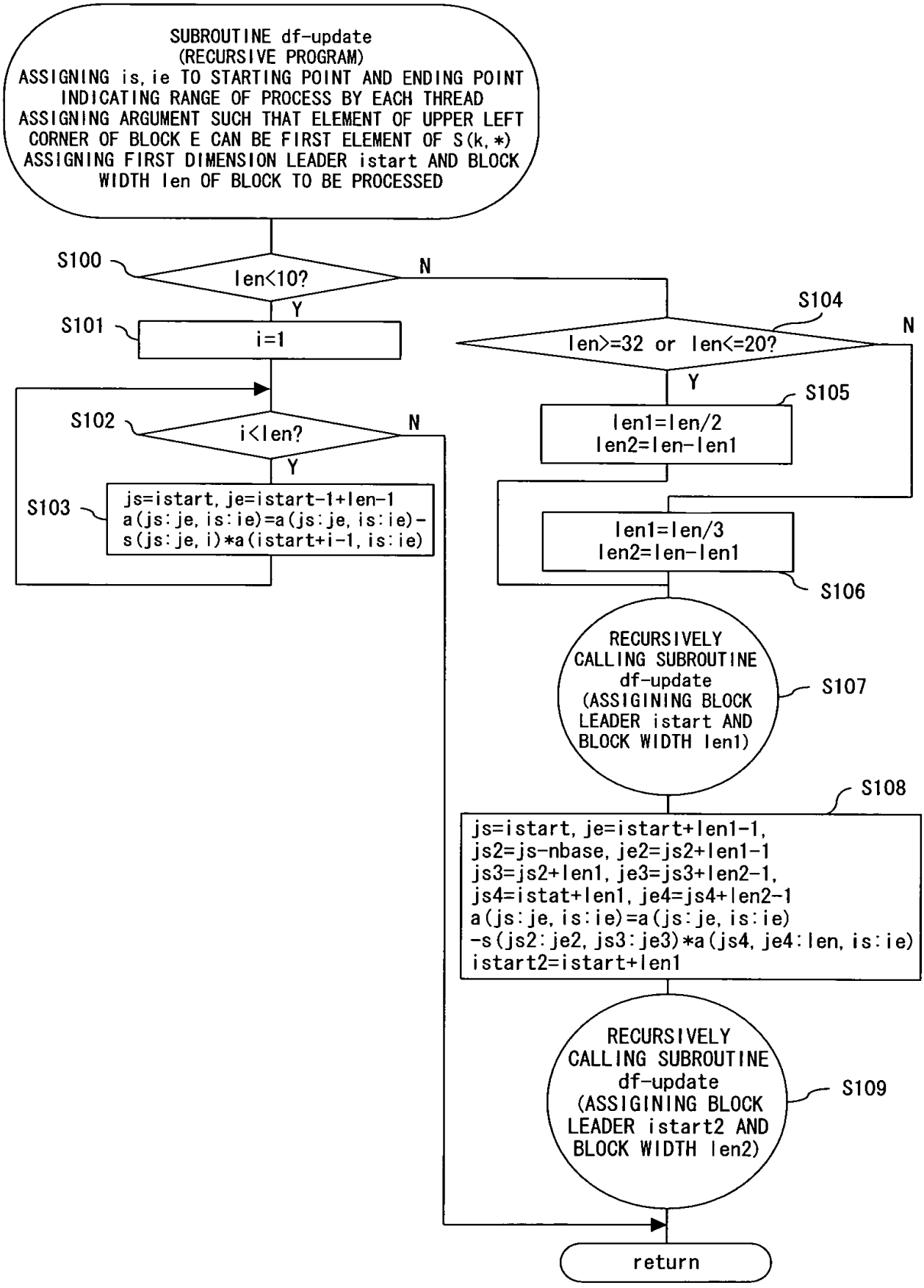

SUBROUTINE df-update
(RECURSIVE PROGRAM)
ASSIGNING is, ie TO STARTING POINT AND ENDING POINT INDICATING RANGE OF PROCESS BY EACH THREAD
ASSIGNING ARGUMENT SUCH THAT ELEMENT OF UPPER LEFT CORNER OF BLOCK E CAN BE FIRST ELEMENT OF S(k, *)
ASSIGNING FIRST DIMENSION LEADER istart AND BLOCK WIDTH len OF BLOCK TO BE PROCESSED
S100
len<10?
N
Y
S101
i=1
S102
i<len?
N
Y
S103
js=istart, je=istart-1+len-1
a(js:je, is:ie)=a(js:je, is:ie)-
s(js:je, i)*a(istart+i-1, is:ie)
S104
len>=32 or len<=20?
N
Y
S105
len1=len/2
len2=len-len1
len1=len/3
len2=len-len1
S106
RECURSIVELY CALLING SUBROUTINE df-update (ASSIGINING BLOCK LEADER istart AND BLOCK WIDTH len1)
S107
S108
js=istart, je=istart+len1-1,
js2=js-nbase, je2=js2+len1-1
js3=js2+len1, je3=js3+len2-1,
js4=istat+len1, je4=js4+len2-1
a(js:je, is:ie)=a(js:je, is:ie)
-s(js2:je2, js3:je3)*a(js4, je4:len, is:ie)
istart2=istart+len1
RECURSIVELY CALLING SUBROUTINE df-update (ASSIGINING BLOCK LEADER istart2 AND BLOCK WIDTH len2)
S109
return

FIG. 26

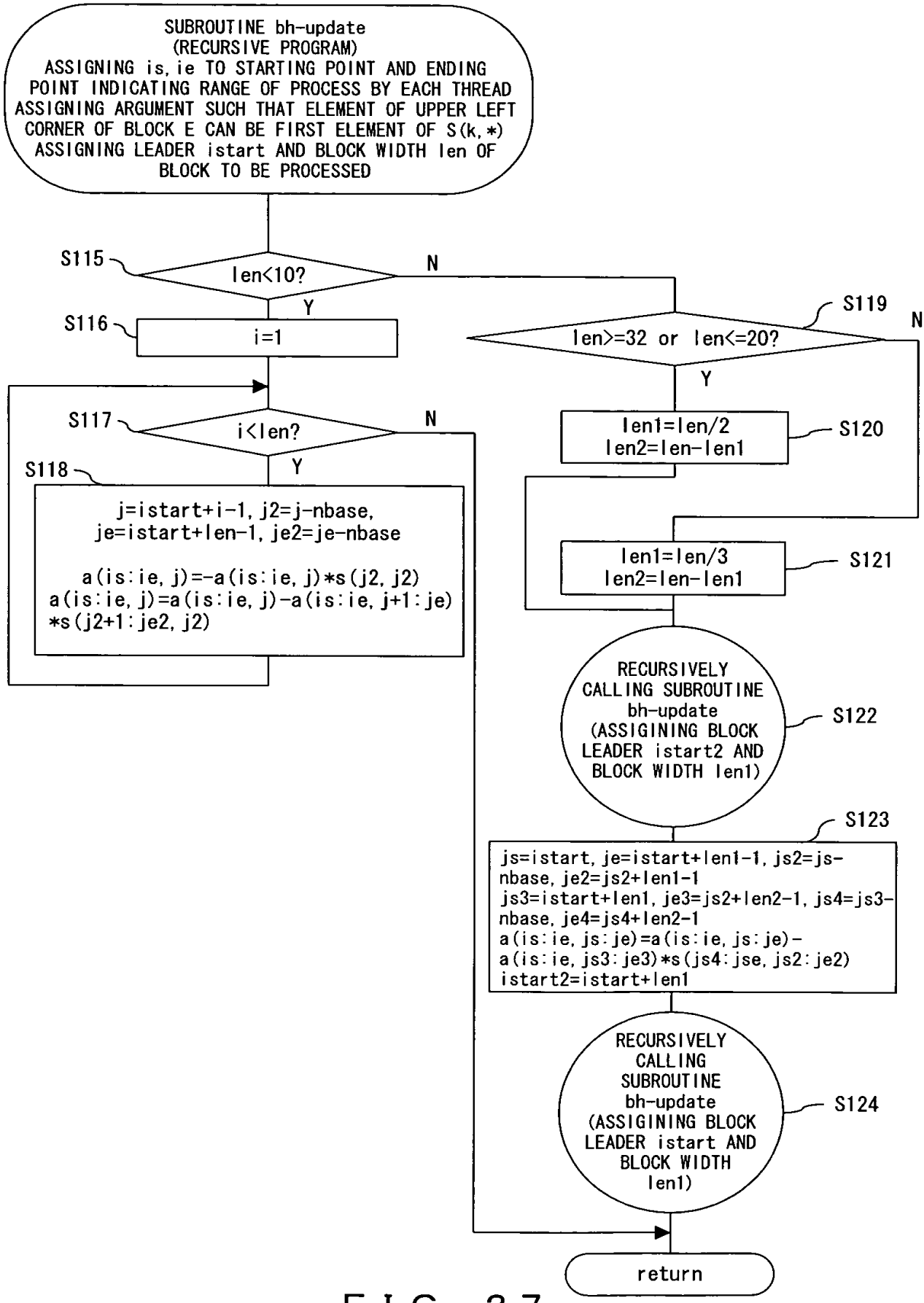

SUBROUTINE bh-update
(RECURSIVE PROGRAM)
ASSIGNING is, ie TO STARTING POINT AND ENDING POINT INDICATING RANGE OF PROCESS BY EACH THREAD
ASSIGNING ARGUMENT SUCH THAT ELEMENT OF UPPER LEFT CORNER OF BLOCK E CAN BE FIRST ELEMENT OF S(k,*)
ASSIGNING LEADER istart AND BLOCK WIDTH len OF BLOCK TO BE PROCESSED
S115
len<10?
N
Y
S116
i=1
S117
i<len?
N
Y
S118
j=istart+i-1, j2=j-nbase,
je=istart+len-1, je2=je-nbase
a(is:ie, j)=-a(is:ie, j)*s(j2, j2)
a(is:ie, j)=a(is:ie, j)-a(is:ie, j+1:je)
*s(j2+1:je2, j2)
S119
len>=32 or len<=20?
N
Y
len1=len/2
len2=len-len1
S120
len1=len/3
len2=len-len1
S121
RECURSIVELY CALLING SUBROUTINE bh-update (ASSIGINING BLOCK LEADER istart2 AND BLOCK WIDTH len1)
S122
S123
js=istart, je=istart+len1-1, js2=js-nbase, je2=js2+len1-1
js3=istart+len1, je3=js2+len2-1, js4=js3-nbase, je4=js4+len2-1
a(is:ie, js:je)=a(is:ie, js:je)-
a(is:ie, js3:je3)*s(js4:jse, js2:je2)
istart2=istart+len1
RECURSIVELY CALLING SUBROUTINE bh-update (ASSIGINING BLOCK LEADER istart AND BLOCK WIDTH len1)
S124
return

FIG. 27

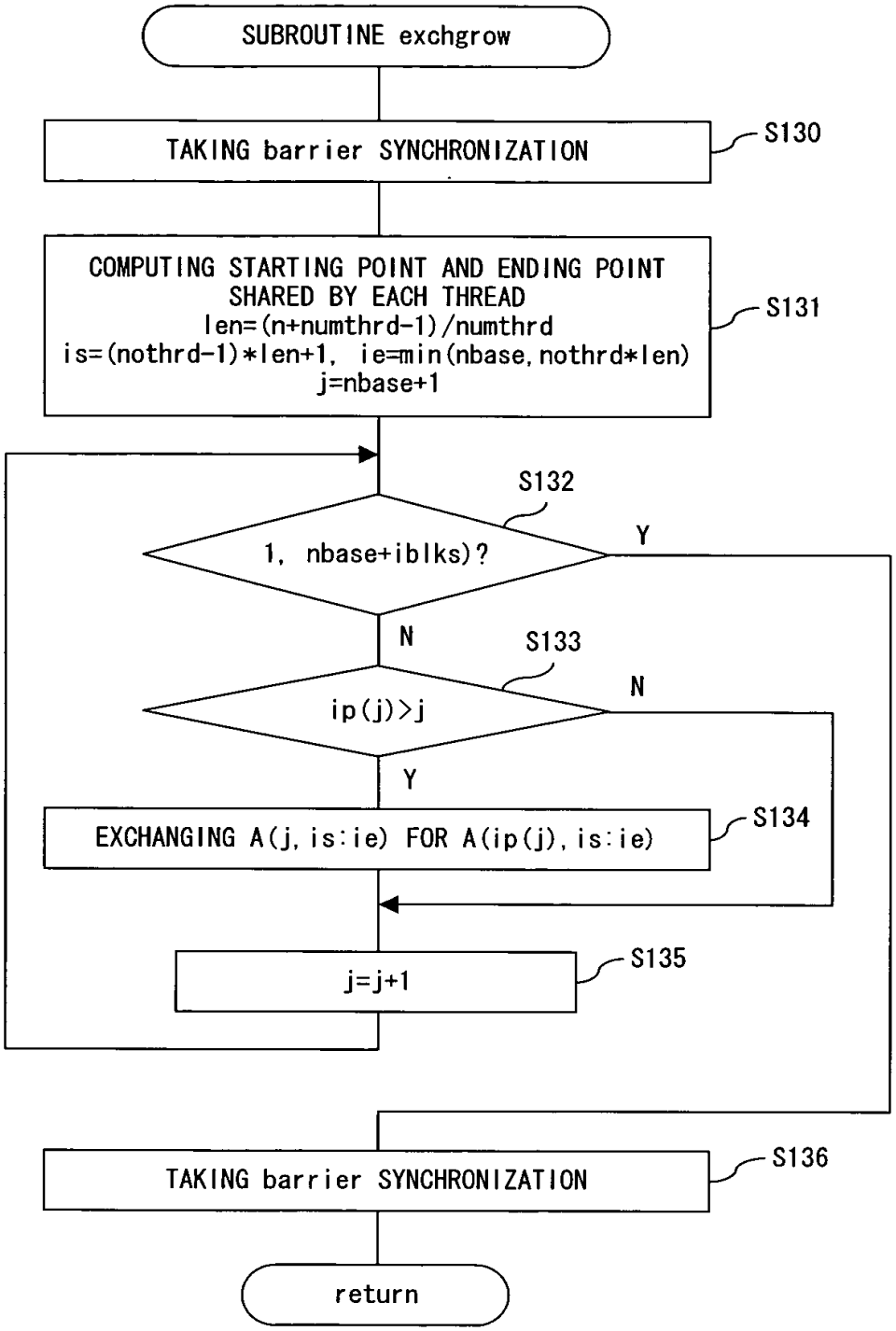

SUBROUTINE exchgrow
TAKING barrier SYNCHRONIZATION
S130
COMPUTING STARTING POINT AND ENDING POINT SHARED BY EACH THREAD
len=(n+numthrd-1)/numthrd
is=(nothrd-1)*len+1, ie=min(nbase,nothrd*len)
j=nbase+1
S131
S132
1, nbase+iblks)?
Y
N
S133
ip(j)>j
N
Y
EXCHANGING A(j,is:ie) FOR A(ip(j),is:ie)
S134
j=j+1
S135
TAKING barrier SYNCHRONIZATION
S136
return

FIG. 28

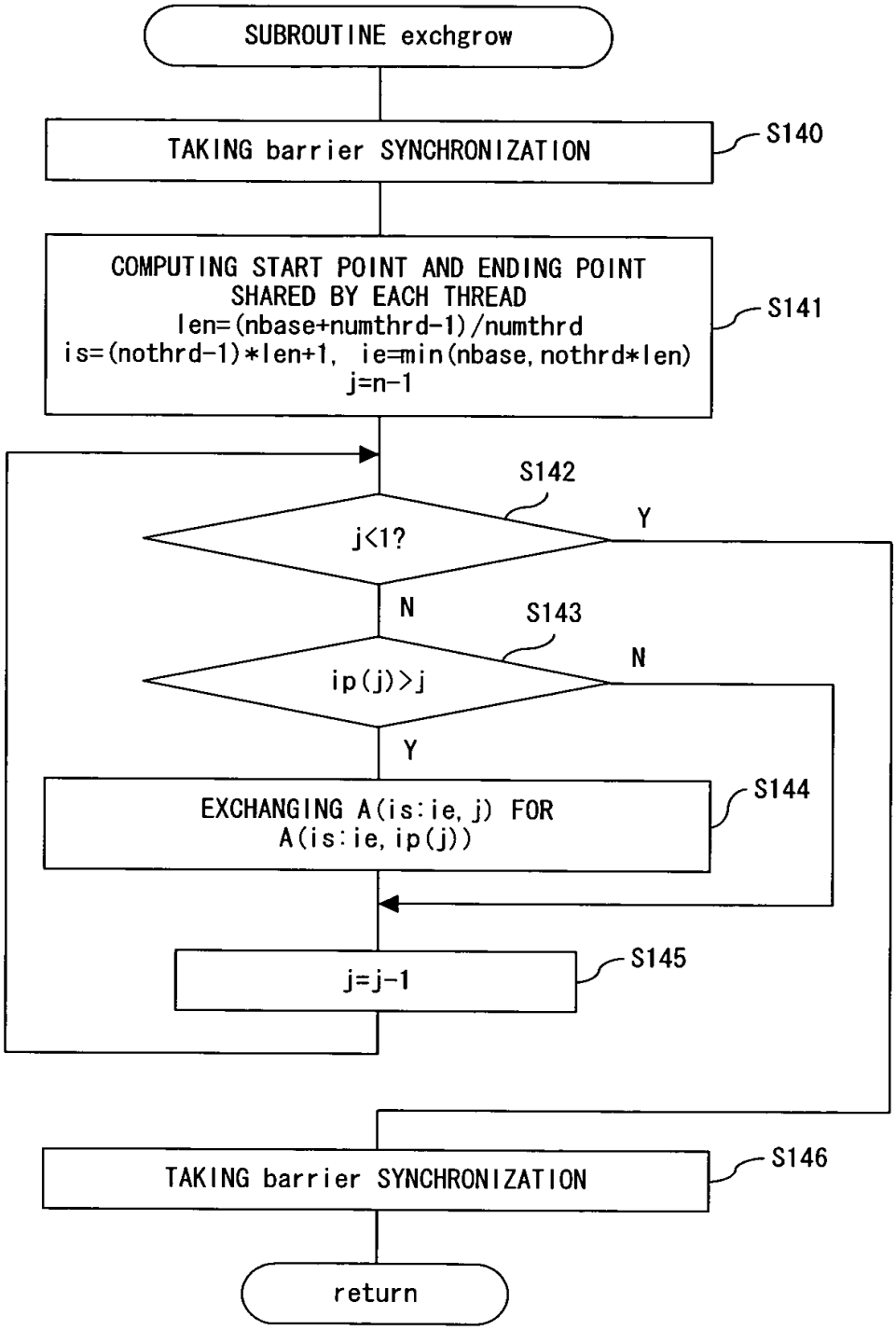

SUBROUTINE exchgrow
TAKING barrier SYNCHRONIZATION
S140
COMPUTING START POINT AND ENDING POINT SHARED BY EACH THREAD
len=(nbase+numthrd-1)/numthrd
is=(nothrd-1)*len+1, ie=min(nbase, nothrd*len)
j=n-1
S141
S142
j<1?
Y
N
S143
ip(j)>j
N
Y
EXCHANGING A(is:ie, j) FOR A(is:ie, ip(j))
S144
j=j-1
S145
TAKING barrier SYNCHRONIZATION
S146
return

FIG. 29